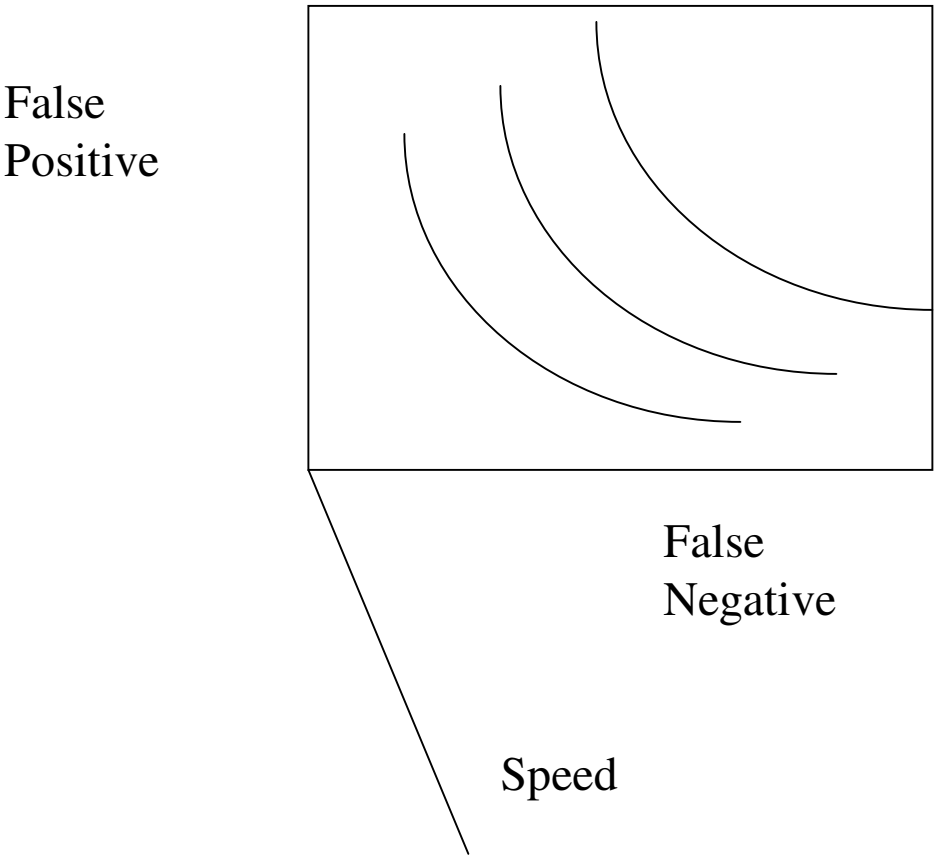
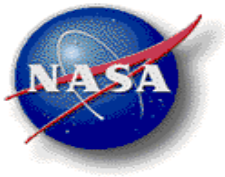
Analytic Verification for Autonomy

Guillaume Brat, Allen Goldberg, Klaus Havelund, Arnaud Venet
ASE Group
NASA Ames

Dimensions of Improvement in Autonomy Program Analysis



DS1 Deadlock

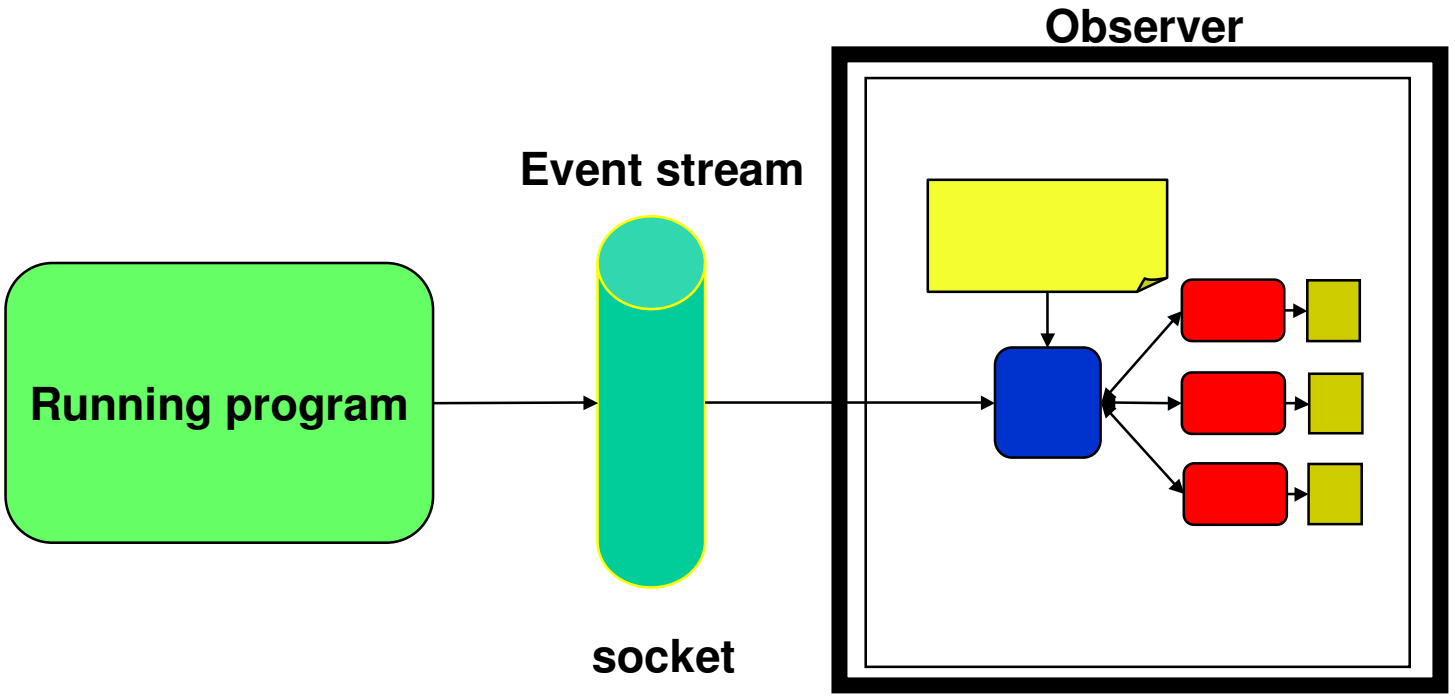


Data race
caused by
Missing critical section
caused
Deadlock

Similar pattern to the one that
Was found using model checking

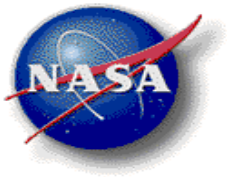
```
(loop
  (when
    (or (/= count (esl::event-count event1))
        (warp-safe (wait-for-event event1)))
    (setf count (esl::event-count event1))
    (signal-event event2)))
```

Could have been found with data race analysis


Observer
Event stream
Running program
socket

Runtime Verification

Testable Concurrency Analysis



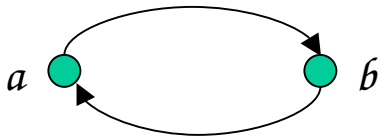
- *Deadlock and data race potentials.*
- *Turning such properties into testable properties.*
  - *High chance of finding errors with few runs*
- *Standard lock set algorithm:*

```
T1:              T2:

lock(a);         lock(b);
lock(b);         lock(a);
```

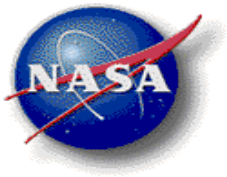
Execute program

Deadlock:
Graph contains cycle



Runtime Verification

Testable Concurrency Analysis



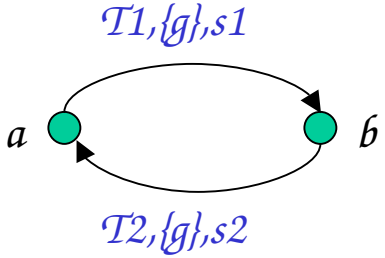
- *Standard algorithm yields false positives*
- *Remove false positives by annotating graph*
  - *Lock hierarchy, segments, threads*
- *Implemented and applied to K9 (35,000 LOC) and ACS - removes false positives*

```
T1:                T2:

lock(g);           lock(g);
lock(a);           lock(b);
lock(b);           lock(a);
start(T2);
```

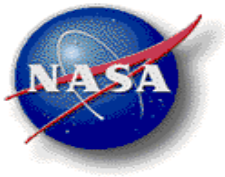
No deadlock:
- *{g} and {g} overlaps*
- *s1 must execute before s2*

Execute program



Runtime Verification

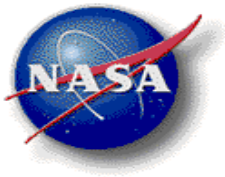
Requirements Checking with Temporal Logic



- Requirements formulated in temporal logic.
- Executing program is monitored during execution.
- Logic must be expressive enough to capture interesting properties:
  - Ordering of events: B follows A.
  - Real-time properties: B follows A within 3 seconds
  - Data properties: B(y) follows A(x) where R(x,y)
- Logics implemented using rewriting system (Maude).
- Case studies done on K9 Rover (ARC) and DS1 Fault Protection System (JPL).

Runtime Verification

Checking Temporal Logic Efficiently



$[](a \rightarrow \langle\rangle b)$

- First Semantics:
  - 100 events : 30 ms (74 K rewrites)
  - 1,000 events : 3 sec (7,3 million rewrites)
  - 10,000 events : > 10 hours (did not terminate over night)
- Second Semantics:
  - <= 10,000 events : << 1 sec
  - 100,000 events : <= 3 sec
  - 100 million events : 1,500 sec (4,9 billion rewrites)
- Second Semantics with "Memo":
  - 100 million events : 185 seconds (230 million rewrites)

Conclusion: an algebraic specification environment such as Maude can be used not only for prototyping but also for final implementation.

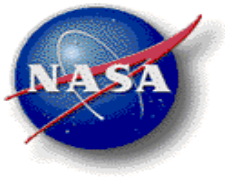
Static Analysis

Guillaume Brat and Arnaud Venet

Static Analysis

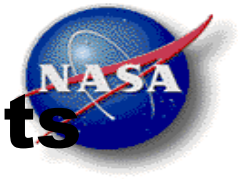
Guillaume Brat and Arnaud Venet

Static Analysis



- Static analysis verifies properties on code without executing it, e.g.,
  - Type (assertion) checking
  - Run-time errors (arithmetic under/overflow, out-of-bound accesses, non-initialized variables/pointers)
- Research:
  - Apply PolySpace, an advanced commercial tool based on abstract interpretation, on NASA mission code
  - Identify technical gaps
  - Establish a research plan to address these gaps

PolySpace: Experimental Results



Project	MPF	ISS	K9 Rover
Language	C	C	C++ => C
Size	200KLocs	40KLocs	35KLocs
Maturity	Stable	Untested	Prototype
Modules	ACS+EDL	HLRC	Executive
Max Size	25KLocs	17KLocs	3.2KLocs
Errors	1 NIV	3OBAI 3OVFL	Old: 1/5 (NIV, OVFL) New: 2UNR 1 NIP 1 OVFL

PolySpace Limitations

- Precision:
  - Array cells merge into one
- Scalability: limited by
  - Size (< 20KLocs)
  - Pointer analysis
  - Multithread combinatorics
- Result interpretation
- Usability

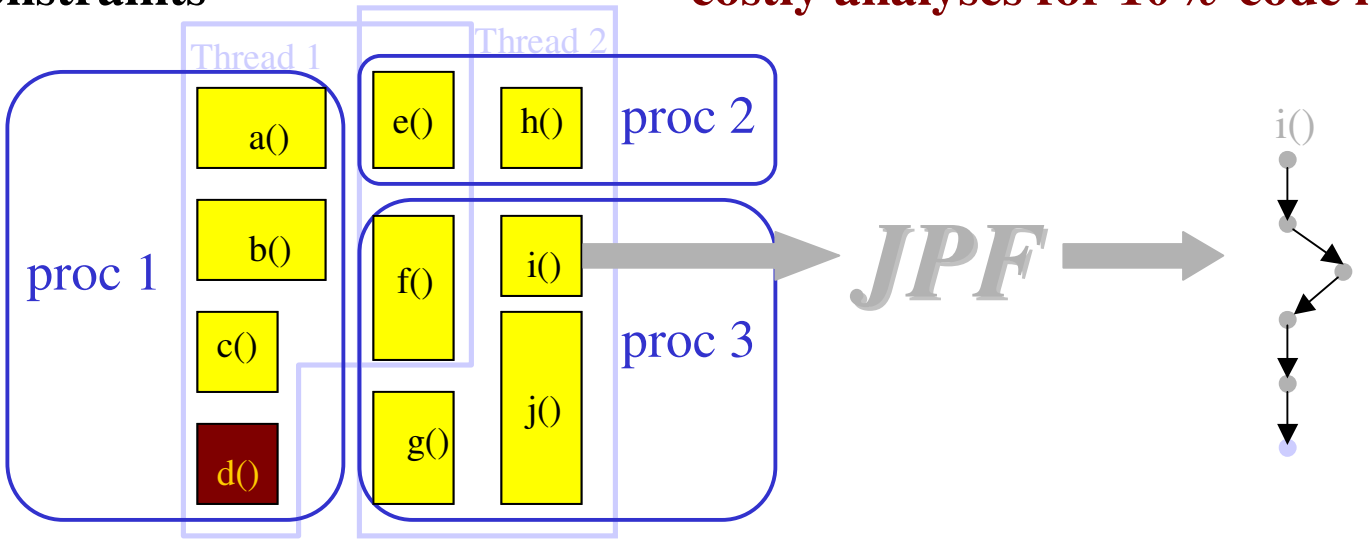
MPF Legacy Coding Practice

- Base data structure: matrix
- Pointers are mainly used
  - to iterate over matrix elements
  - in complex loop structures
- Mostly static data
  - Marginal use of dynamically allocated structures
- Several threads of execution

C Global Surveyor

Specialized pointer analysis
precise for top-level pointers
thread sensitive
Supplement pointer info
with index range constraints

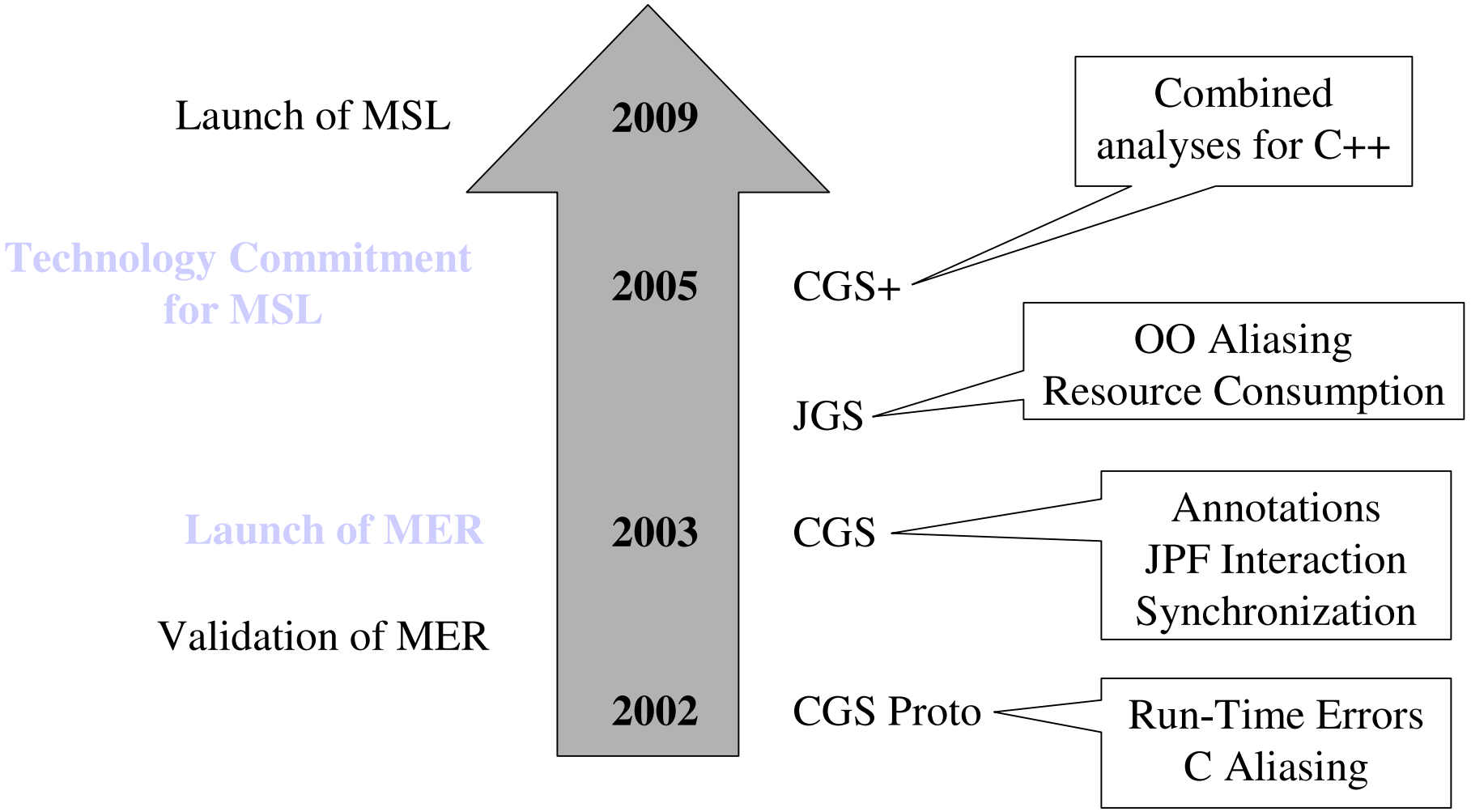
Incremental refinement of analyses
build analyses on top of each other
simple analyses for 90% of code
complex analyses refines simpler ones
costly analyses for 10% code left

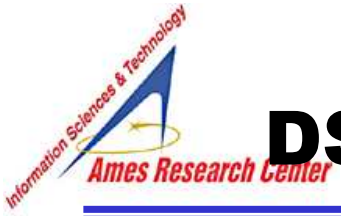


granularity of algorithms is function
context passing:
low overhead w.r.t. computation time
Distributed abstract interpretation

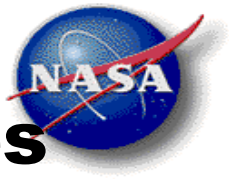
use JPF to generate scenarios
to illustrate certain errors
and to filter false positives
Smart result interpretation

Technology Roadmap





DS1 (MPF legacy) Defect Classes

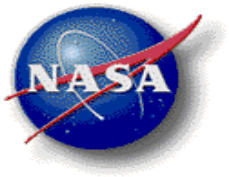


- Concurrency: race conditions, deadlocks
- Misuse: array out-of-bound, pointer mis-assignments
- Initialization: no value, incorrect value
- Assignment: wrong value, type mismatch
- Computation: wrong equation
- Undefined Ops: FP errors (tan(90)), arithmetic (division by zero)
- Omission: case/switch clauses without defaults
- Scoping Confusion: global/local, static/dynamic
- Argument Mismatches: missing args, too many args, wrong types, uninitialized args
- Finiteness: underflow, overflow

Analysis of Mars mission code

Guillaume Brat

Static Analysis of MPF



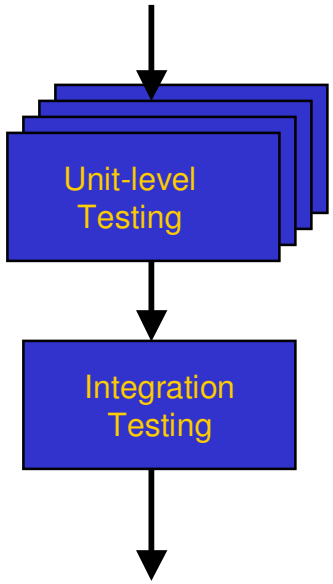
- Goal: apply state-of-the-art static analysis tool to Mars PathFinder
  - Assess current commercial capabilities
  - Assess potential application to MER
  - Identify technological gaps
- Technique: Abstract Interpretation
  - Compute a superset of the range of each variable
  - Cover all possible paths without executing the code
  - Check all computed ranges against the domain of definition of each risky operation

PolySpace C-Verifier

Shown: static analysis based on abstract interpretation finds all possible run-time errors (e.g., out-of-bound array accesses, dereferencing through null pointers, illegal type conversions, invalid arithmetic conversions, overflow/underflow, non-initialized variables, and access conflicts for unprotected shared data), which are difficult to detect through conventional testing.

Conventional Testing

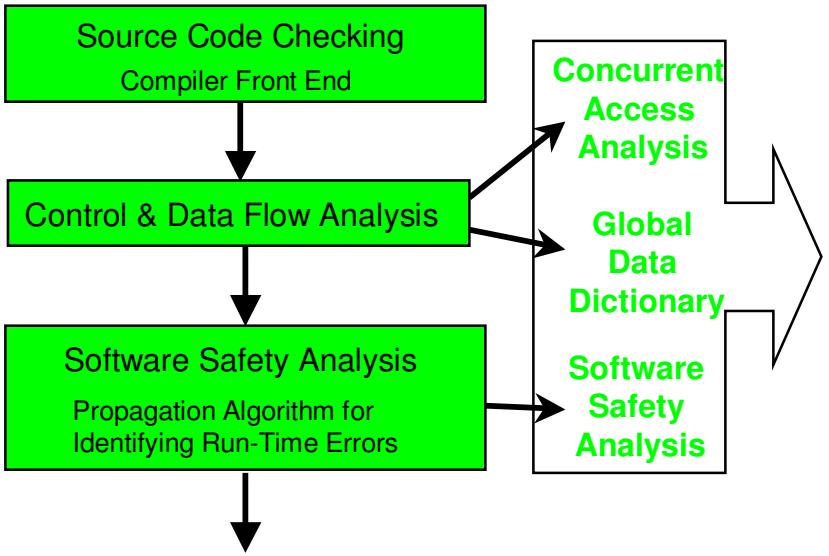
Test cases & drivers



Partial Error Coverage

Sophisticated Static Analysis

No input cases! No input drivers!



Total Error Coverage

Concurrent Access Analysis

Global Data Dictionary

Software Safety Analysis

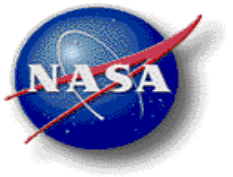
Simple run-time error reporting

```
    p = x - 0.75;
    y = sqrt (p);
}

/* unreachable or dead code
void unr () {
    int x = random_int();
    int y = random_int();
        if (x > y) {
            x = x - y;
            if (x < 0) {
```

Analysis time ~ e (error selectivity)

Mars PathFinder Analysis



- 23 modules of stable C code for 200KLocs
  - Focused the analysis on two critical modules
- EDL module was shown to be mature:
  - No red checks in 15KLocs with 3 threads
  - Orange checks were dismissed by manually inspecting separate initialization code
- ACS module was also fairly mature:
  - Only 1 red check (NIV) in 25KLocs with 3 threads
  - Not critical, but prevented optimization code to execute
  - Error is of the same class as the one that caused the crash of Mars Polar Lander